<http://hyperledger-fabric.readthedocs.io/en/latest/build_network.html>

/\*

\* Licensed to the Apache Software Foundation (ASF) under one

\* or more contributor license agreements. See the NOTICE file

\* distributed with this work for additional information

\* regarding copyright ownership. The ASF licenses this file

\* to you under the Apache License, Version 2.0 (the

\* "License"); you may not use this file except in compliance

\* with the License. You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing,

\* software distributed under the License is distributed on an

\* "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

\* KIND, either express or implied. See the License for the

\* specific language governing permissions and limitations

\* under the License.

\*/

/\*

\* The sample smart contract for documentation topic:

\* Writing Your First Blockchain Application

\*/

package main

/\* Imports

\* 4 utility libraries for formatting, handling bytes, reading and writing JSON, and string manipulation

\* 2 specific Hyperledger Fabric specific libraries for Smart Contracts

\*/

import (

"bytes"

"encoding/json"

"fmt"

"strconv"

"github.com/hyperledger/fabric/core/chaincode/shim"

sc "github.com/hyperledger/fabric/protos/peer"

)

// Define the Smart Contract structure

type SmartContract struct {

}

// Define the car structure, with 4 properties. Structure tags are used by encoding/json library

type Car struct {

Make string `json:"make"`

Model string `json:"model"`

Colour string `json:"colour"`

Owner string `json:"owner"`

}

/\*

\* The Init method is called when the Smart Contract "fabcar" is instantiated by the blockchain network

\* Best practice is to have any Ledger initialization in separate function -- see initLedger()

\*/

func (s \*SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {

return shim.Success(nil)

}

/\*

\* The Invoke method is called as a result of an application request to run the Smart Contract "fabcar"

\* The calling application program has also specified the particular smart contract function to be called, with arguments

\*/

func (s \*SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {

// Retrieve the requested Smart Contract function and arguments

function, args := APIstub.GetFunctionAndParameters()

// Route to the appropriate handler function to interact with the ledger appropriately

if function == "queryCar" {

return s.queryCar(APIstub, args)

} else if function == "initLedger" {

return s.initLedger(APIstub)

} else if function == "createCar" {

return s.createCar(APIstub, args)

} else if function == "queryAllCars" {

return s.queryAllCars(APIstub)

} else if function == "changeCarOwner" {

return s.changeCarOwner(APIstub, args)

}

return shim.Error("Invalid Smart Contract function name.")

}

func (s \*SmartContract) queryCar(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

if len(args) != 1 {

return shim.Error("Incorrect number of arguments. Expecting 1")

}

carAsBytes, \_ := APIstub.GetState(args[0])

return shim.Success(carAsBytes)

}

func (s \*SmartContract) initLedger(APIstub shim.ChaincodeStubInterface) sc.Response {

cars := []Car{

Car{Make: "Toyota", Model: "Prius", Colour: "blue", Owner: "Tomoko"},

Car{Make: "Ford", Model: "Mustang", Colour: "red", Owner: "Brad"},

Car{Make: "Hyundai", Model: "Tucson", Colour: "green", Owner: "Jin Soo"},

Car{Make: "Volkswagen", Model: "Passat", Colour: "yellow", Owner: "Max"},

Car{Make: "Tesla", Model: "S", Colour: "black", Owner: "Adriana"},

Car{Make: "Peugeot", Model: "205", Colour: "purple", Owner: "Michel"},

Car{Make: "Chery", Model: "S22L", Colour: "white", Owner: "Aarav"},

Car{Make: "Fiat", Model: "Punto", Colour: "violet", Owner: "Pari"},

Car{Make: "Tata", Model: "Nano", Colour: "indigo", Owner: "Valeria"},

Car{Make: "Holden", Model: "Barina", Colour: "brown", Owner: "Shotaro"},

}

i := 0

for i < len(cars) {

fmt.Println("i is ", i)

carAsBytes, \_ := json.Marshal(cars[i])

APIstub.PutState("CAR"+strconv.Itoa(i), carAsBytes)

fmt.Println("Added", cars[i])

i = i + 1

}

return shim.Success(nil)

}

func (s \*SmartContract) createCar(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

if len(args) != 5 {

return shim.Error("Incorrect number of arguments. Expecting 5")

}

var car = Car{Make: args[1], Model: args[2], Colour: args[3], Owner: args[4]}

carAsBytes, \_ := json.Marshal(car)

APIstub.PutState(args[0], carAsBytes)

return shim.Success(nil)

}

func (s \*SmartContract) queryAllCars(APIstub shim.ChaincodeStubInterface) sc.Response {

startKey := "CAR0"

endKey := "CAR999"

resultsIterator, err := APIstub.GetStateByRange(startKey, endKey)

if err != nil {

return shim.Error(err.Error())

}

defer resultsIterator.Close()

// buffer is a JSON array containing QueryResults

var buffer bytes.Buffer

buffer.WriteString("[")

bArrayMemberAlreadyWritten := false

for resultsIterator.HasNext() {

queryResponse, err := resultsIterator.Next()

if err != nil {

return shim.Error(err.Error())

}

// Add a comma before array members, suppress it for the first array member

if bArrayMemberAlreadyWritten == true {

buffer.WriteString(",")

}

buffer.WriteString("{\"Key\":")

buffer.WriteString("\"")

buffer.WriteString(queryResponse.Key)

buffer.WriteString("\"")

buffer.WriteString(", \"Record\":")

// Record is a JSON object, so we write as-is

buffer.WriteString(string(queryResponse.Value))

buffer.WriteString("}")

bArrayMemberAlreadyWritten = true

}

buffer.WriteString("]")

fmt.Printf("- queryAllCars:\n%s\n", buffer.String())

return shim.Success(buffer.Bytes())

}

func (s \*SmartContract) changeCarOwner(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

if len(args) != 2 {

return shim.Error("Incorrect number of arguments. Expecting 2")

}

carAsBytes, \_ := APIstub.GetState(args[0])

car := Car{}

json.Unmarshal(carAsBytes, &car)

car.Owner = args[1]

carAsBytes, \_ = json.Marshal(car)

APIstub.PutState(args[0], carAsBytes)

return shim.Success(nil)

}

// The main function is only relevant in unit test mode. Only included here for completeness.

func main() {

// Create a new Smart Contract

err := shim.Start(new(SmartContract))

if err != nil {

fmt.Printf("Error creating new Smart Contract: %s", err)

}

}